188097 515-63
108.

Distributed Intelligence for Supervisory Control

W.J. Wolfe and S.D. Raney
Martin Marietta Denver Aerospace
Denver, CO 80201

MI 411300

1. ABSTRACT

Supervisory control systems must deal with various types of "intelligence" distributed throughout the layers of control. Typical layers are real-time servo control, off-line planning and reasoning subsystems and finally, the human operator. Design methodologies must account for the fact that the majority of the intelligence will reside with the human operator. ~~This paper focuses~~ on hierarchical decompositions and feedback loops as conceptual building blocks that provide a common ground for man-machine interaction. Examples of types of parallelism and parallel implementation on several classes of computer architecture are also discussed.

Are discussed.

2. INTRODUCTION

Designing systems that involve a lot of interaction with humans is a difficult task. Although humans are intelligent they are also unpredictable. It is easier to conceive of completely "autonomous" systems, requiring little man-machine interaction. But with such systems human operators run the risk of losing control and at some point the operator may want to redirect, assist, troubleshoot or enhance the system. Thus, it is desirable to be operating somewhere in the middle ground, where systems are designed to be autonomous only in the sense that they are part of a larger system that provides a means of interaction with humans. The concept of "supervisory control" provides such a framework.

2.1 General System Design Goal

Put in simple terms, it is desirable to have systems that are reliable, flexible and expandable (Figure 1). Reliability implies that the system rarely fails and that there is a quick fix for failures such as plugging in a new readily available module. Along with prior testing and evaluation is the development of run-time aids such as fault detection, isolation and diagnosis systems, all of which improve reliability. A flexible system is one that can perform many different functions or tasks and can operate in diverse environments. Such a system must consider many alternatives and be able to "understand" its environment. After a system is fielded it is invariably discovered that new capabilities are desired or required. An expandable system can easily incorporate new functions and more knowledge.

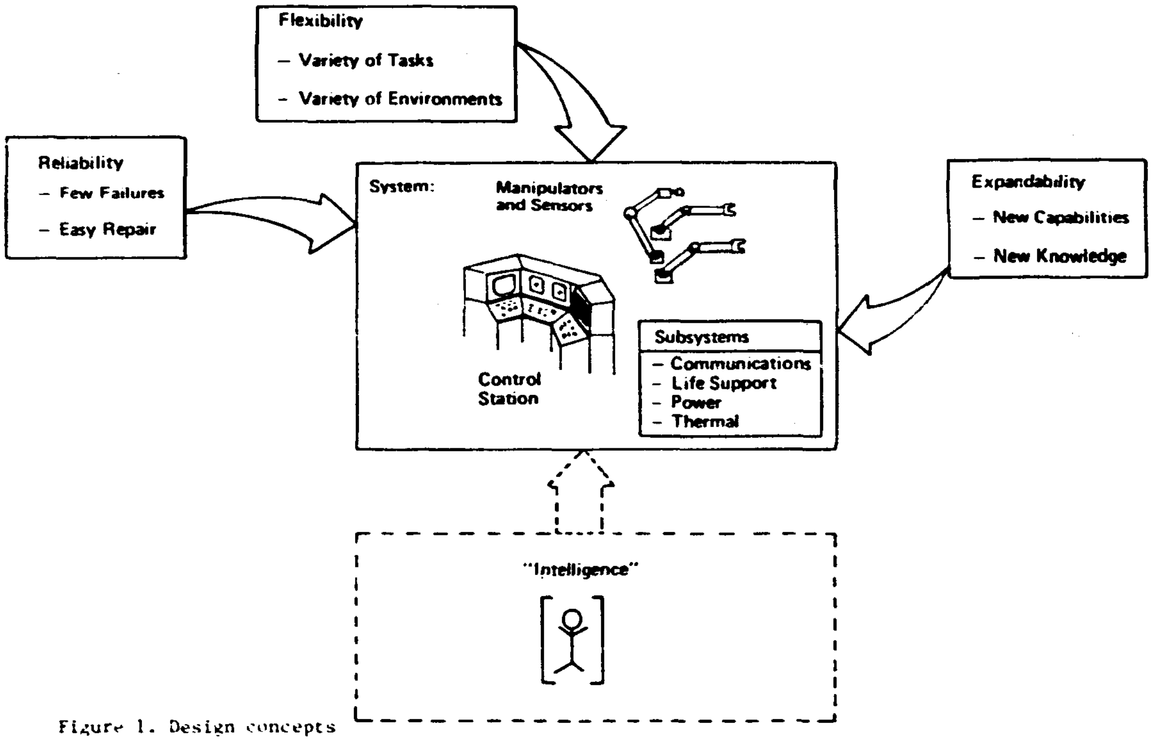


Figure 1. Design concepts

Many of these design goals boil down to providing the system with "intelligence". Although there has been much progress in the development of systems that display limited forms of intelligence,it is often the case that when intelligence is required human operators must be incorporated. Therefore, the development of complex systems now and in the near future will rely heavily on human operators (Figure 2).

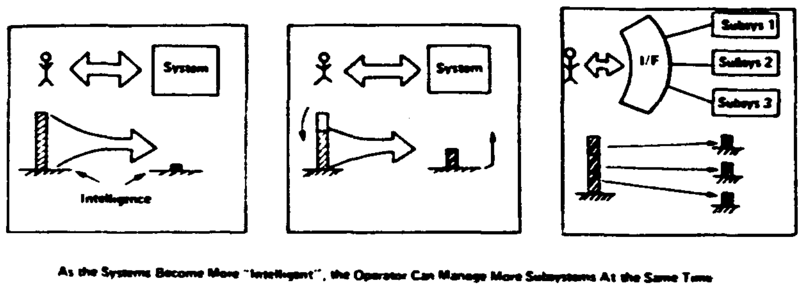

System
Intelligence
System
I/F
Subsys 1
Subsys 2
Subsys 3
As the Systems Become More "Intelligent", the Operator Can Manage More Subsystems At the Same Time

Figure 2. A definition of "supervisory control"

2.2 Supervisory Control

To effectively operate a system, a human must focus attention and can only expend a certain amount of energy. The more autonomous the system, the less attention required from the human operator. If the goal is to have less and less for the human operator to do the human would ultimately become a "watchman" asleep until an alarm sounds. Unfortunately, human expertise should be used more efficiently. Thus, the proper direction to move in is that as a particular system becomes "smarter", the operator devotes less attention to it and devotes increasingly more attention to a second system, and so on. From this perspective, the operator has various, possibly autonomous, subsystems under his control as he orchestrates the performance of the overall system.

2.3 Amplifiers of Human Capabilities

Machines, in general, enhance human capabilities. Machines give strength, reach and dexterity not normally possessed by humans; they can project presence into areas that would not normally be entered; they allow detection of things that are beyond the capability of human senses and they can enhance our analytical ability by rapidly analyzing data and summarizing the results (Figure 3). In this sense machines act as "amplifiers" of our capabilities, including the amplification of our intelligence. From this point of view, the design of a complex system is seen in terms of a centrally placed human operator - the source of intelligence - supported by advanced man-machine interfaces that provide an array of "tools". The tools are not intended to replace man, but to make it easier for him to perform difficult tasks.

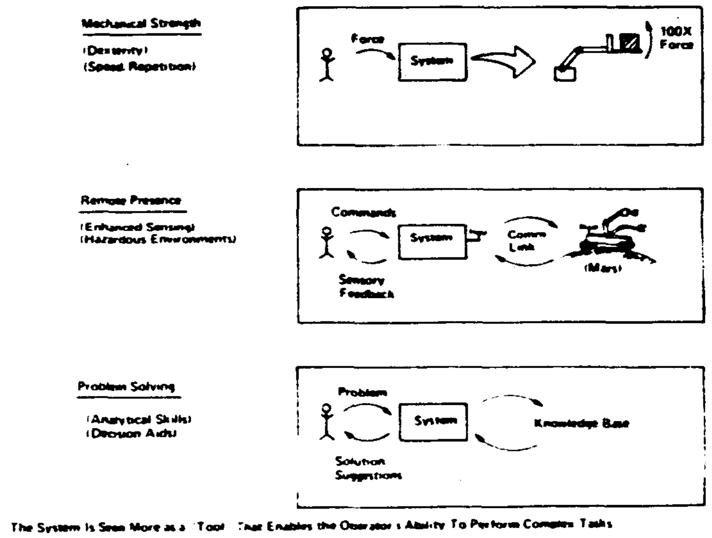


Figure 3. The system acts as an "amplifier" of human capabilities

3. LEVELS AND LOOPS

In seeking a common ground for both man and machine, two design techniques stand out: hierarchical decompositions and feedback loops (Figure 4). Although these two concepts are not mutually exclusive, they have contrasting properties, *emphasizing different aspects of the system under scrutiny. Just about every system has elements of both.* In fact, the same system might be described in terms of a hierarchy or a particular feedback loop, depending on the emphasis desired.

First we will discuss hierarchies.

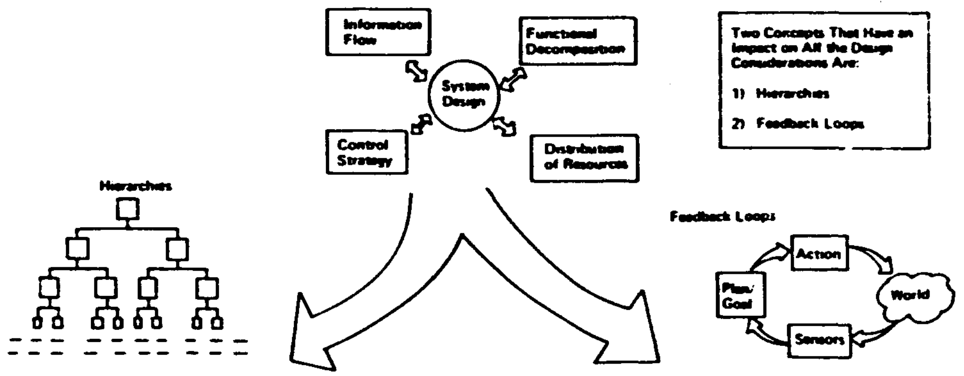


Figure 4. System design methodologies-levels and loops

3.1 Hierarchies

Hierarchies are pervasive, appearing in societal structures such as the military, corporations, churches, and in more abstract areas such as general problem solving (Figure 5). Some of the simplest hierarchies appear in knowledge representation, resulting from natural taxonomies such as "part-of", "is-a" or "kind-of". Similarly, spatial decompositions such as topographical maps or geometric descriptions of a robot environment treated at various levels of resolution provide examples of hierarchical decomposition. Approaches to complex problems such as speech and image understanding invariably involve levels, with the lowest ones associated with "primitives", the middle ones with various patterns and the highest ones with symbolic interpretations. In [1], the organization of functions in a robotic environment such as an automated factory is described as a hierarchy, with high levels related to planning and reasoning about the task and low levels identified with servo control.

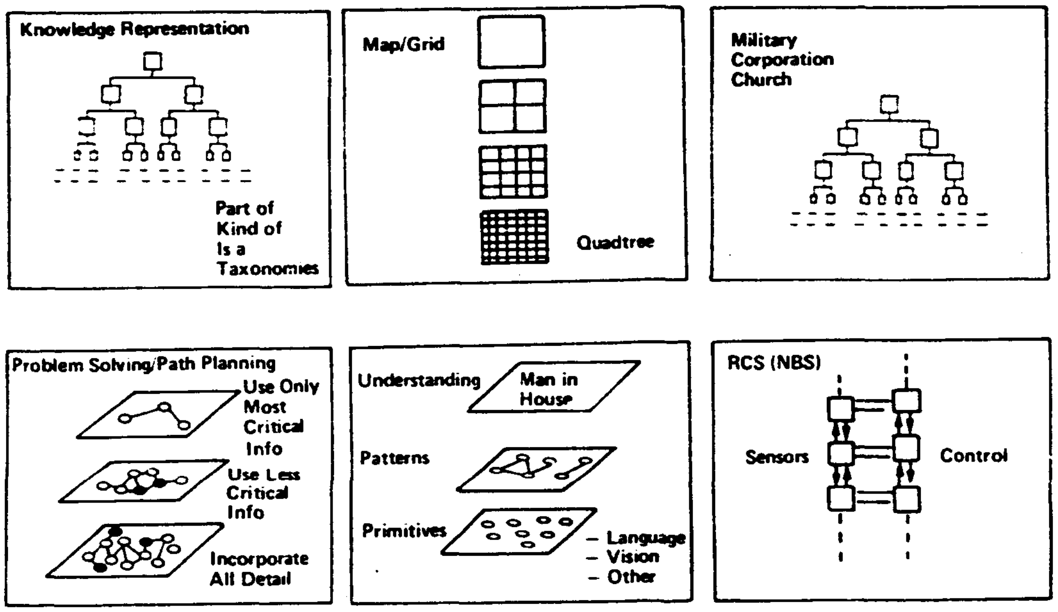


Figure 5. Examples of hierarchies

After looking at several hierarchies some general properties emerge. In more complicated scenarios, such as the decompostion of tasks for a robot, the highest levels are associated with reasoning and planning typically using very general global or abstract knowledge. As a result the highest levels tend to rely on thoughtful, slow, symbolic processing (Figure 6). The overall goals of the system are set at the highest levels, typically considering many alternatives, representing the "goal-driven" aspects of the system. In contrast, the lowest levels are associated with "action", using specific local knowledge. At the lowest levels are numeric, algorithmic processing that results in fast reflex action representing the "data-driven" aspects of the system. The overall hierarchy usually represents a decomposition of functions that is equivalent to the successive reduction of a problem into smaller and easier subproblems.

The dynamics in a hierarchy become apparent when it is realized that information can flow in either direction, creating loops that can straddle several levels (Figure 7). Furthermore, each level must be able to communicate with levels above and below, possibly requiring separate, local, languages, and also possibly swamping the middle levels with too many messages.

A hierarchical approach is not always easy to apply. For complex systems, defining the appropriate levels is quite

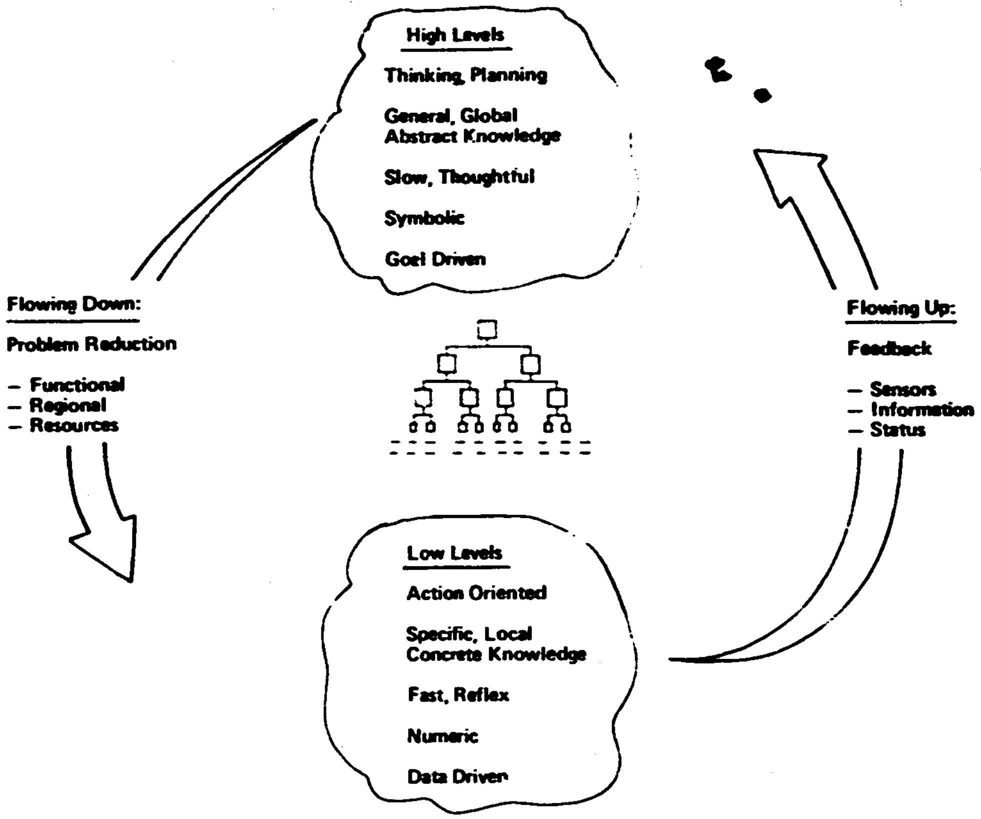


Figure 6. General aspects of hierarchies

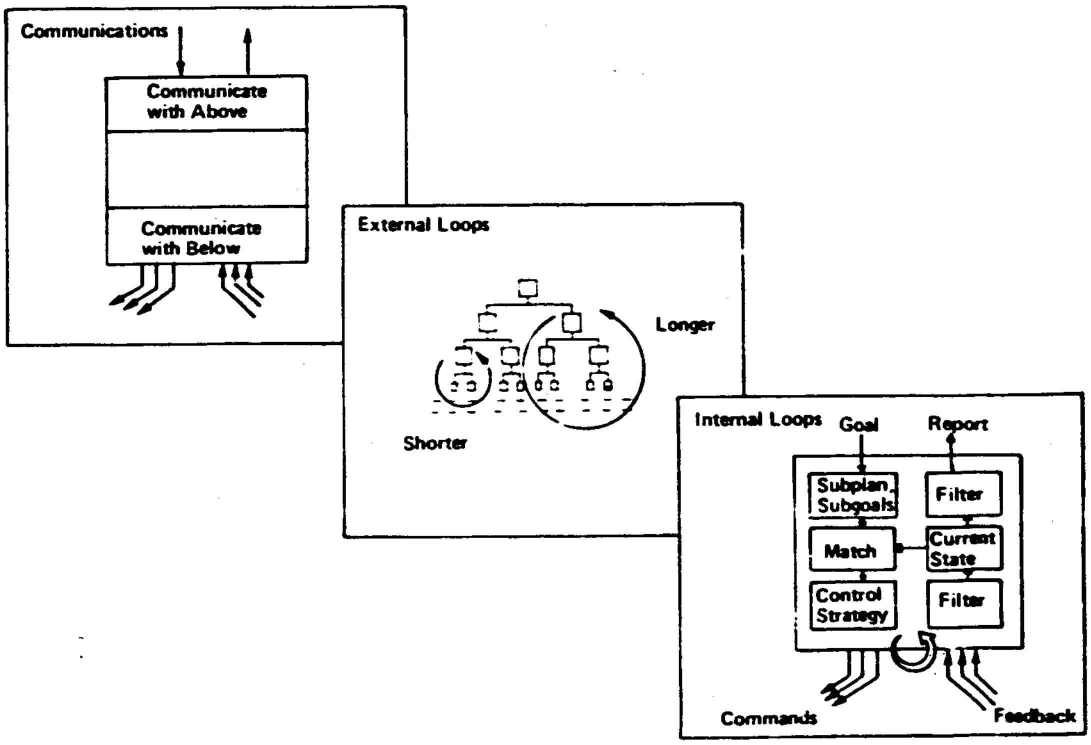
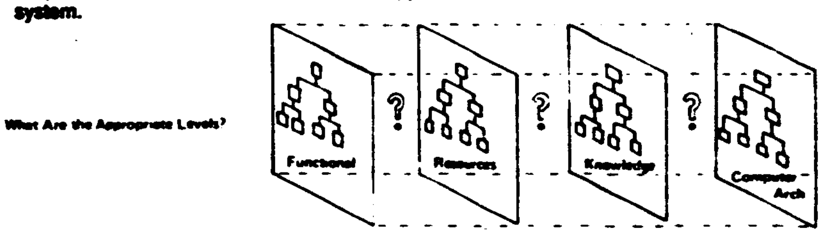


Figure 7. Interlevel communication and loops within hierarchies

complicated, running up against the same difficulties associated with general problem solving and knowledge representation. The problem reduction inherent in the hierarchical decomposition can fall short because of complex interrelations among subproblems. Nevertheless, a hierarchical approach is extremely powerful in its ability to guide the design of a complex system.



- The Levels Are Relatively Easy To Define for Taxonomies, Variable Resolution Maps, Independent Functions (Work Cells), etc
- For Complex Situations, We Are Up Against
  - General Problem Solving Problem Reduction
  - Knowledge Representation

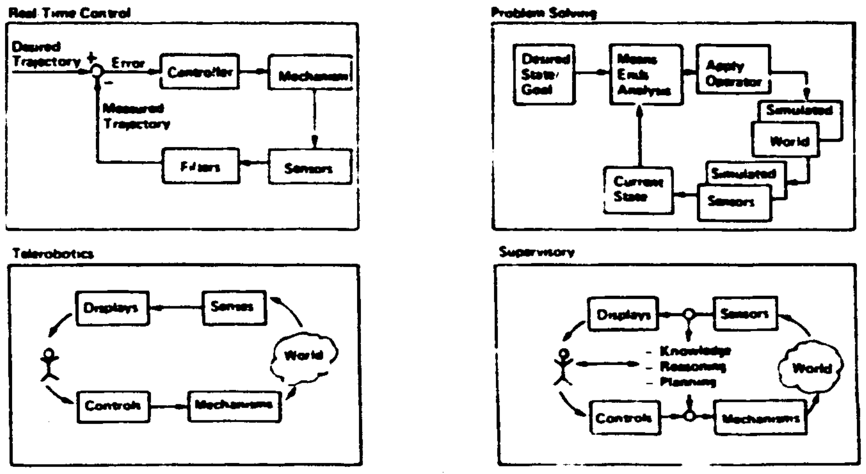
If Levels Are Not Defined Properly, The System Could Easily Get Swamped with Message-Passing Bottlenecks

Figure 8. The challenge of hierarchies

3.2 Feedback Loops

While the use of hierarchies reflects the need to decompose large systems into smaller tractable subsystems, feedback reflects the need to account for uncertainty. If a human operator or an autonomous system had perfect knowledge there would be no need for feedback since things would always go according to plan. Unfortunately, knowledge is often incomplete and erroneous and as a result actions must be followed by some form of checking.

The basic process underlying a feedback loop consists of two steps, repeated until successful : 1) using feedback, either from sensors or other sources of information, compare current state with desired state; 2) based on the results cf the first step, decide on an action that will move the system closer to the desired state. Figure 9 depicts several types of feedback loops.



- Feedback Helps Account for Uncertainty
- The Emphasis Is on Incremental Progress Toward a Goal

Figure 9. Feedback loops

The first step assumes that sufficient information can be gathered from the particular domain of interest (ie. "world") and represented in a form that can be compared with some representation of the goal. For example, a simple servo control loop matches a desired with a measured trajectory to create an error signal for the "controller" which attempts to drive the error to zero by the proper actuation. In this way a relatively simple model of the "world" can be very effective. By focusing on the critical information as represented by the error signal, a rapid response can be easily formulated; but robust behavior (such as adaptation to load changes, collision avoidance and compliant control) is difficult to achieve without detailed dynamic models. That is, in most cases the decision on what to do at each iteration is not obvious. Although the iterative nature of the feedback loop implies incremental or local progress toward the goal, global knowledge must be used as well to avoid dead ends and local extrema.

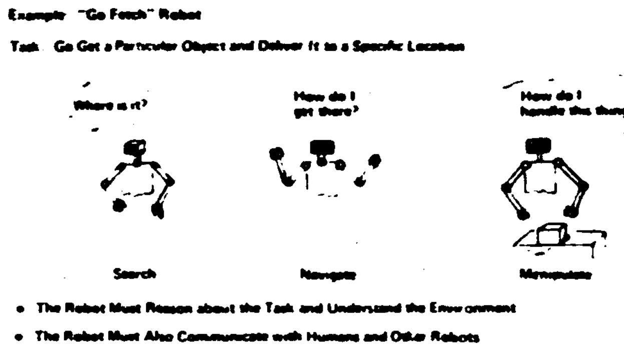
Another basic example is the "means-ends" analysis used as a problem solving paradigm. Here, the representations of states, goals and available operators can be quite abstract, but the approach can be described in terms of a loop. A comparison of the current with the desired state detects a difference that is used to select operators that can reduce the difference. Typically, this produces a search process as various operators are tried out on a simulated "world". Once again, though, one of the major difficulties in applying such a procedure is the need to recognize global contraints at the incremental level.

Although there are various problems in applying feedback loops, it is clear that they are very powerful as a means of isolating and representing the critical aspects of a system, especially when uncertainty plays a major role.

4. Example : A SUPERVISED ROBOT

Our example is a "go-fetch" robot. The point of this example is to demonstrate that simple sounding tasks are tremendously complicated when considered for automation. Although various hierarchies and feedback loops can be readily identified, the interrelationships among them creates complexity that often requires human level intelligence.

Consider the task as beginning with a command to go get a particular object and deliver it to some specified location. The highest levels of a functional hierarchy might break the task into three subtasks : search for the object; navigate in the environment; manipulate the object (Figure 10). One of the basic problems already is that each of these will require an ability to reason about the task and the environment that is difficult to specify, and furthermore, there will be a need for the robot to communicate with the supervisor and possibly other robots.


Example "Go Fetch" Robot
Task: Go Get a Particular Object and Deliver It to a Specific Location
Where is it?
How do I get there?
How do I handle this thing?
Search
Navigate
Manipulate
The Robot Must Reason about the Task and Understand the Environment
The Robot Must Also Communicate with Humans and Other Robots
Figure 10. Typical task

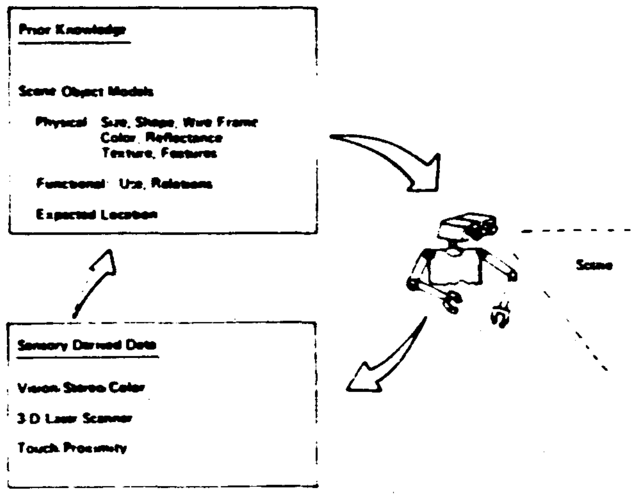


Figure 11. Search

4.1 Search

Unless the environment is extremely constrained, searching for an object will rely primarily on vision (and possibly range images). This sensory data must be processed and compared with internal models of the object's physical properties, functional aspects and expected location. In general, this is known to be a very difficult task, involving hierarchical structures that attempt to organize knowledge driven processes arising from expectations, and the data driven processes arising from the reduction of sensory data into image primitives such as edges, lines and surfaces. Searching for objects will require a significant amount of help from the human supervisor.

4.2 Navigate

Navigation entails the interplay of an interesting combination of information : 1) prior knowledge of the environment, typically described in global terms since the details would be hard to keep track of (for example, a description of the structural components such as the main passageways and known beacons could be available, but small obstacles would have to be discovered); 2) The robot's sensors such as vision and proximity would have to provide the local information about the environment, and use it to avoid obstacles and recognize landmarks; 3) internal motion sensing could be used to give the robot an estimate of its position independent of the external sensors; 4) finally, planning paths that negotiate local difficulties while getting to the goal requires geometric reasoning capability.

In order to successfully navigate, the robot must use all these sources of information, recognizing landmarks, obstacles, tight spots and dead ends while reasoning about progress toward the goal. Once again, a navigation task would be easy in a very structured environment, but in real applications the problem is quite difficult, requiring the human operator's assistance.

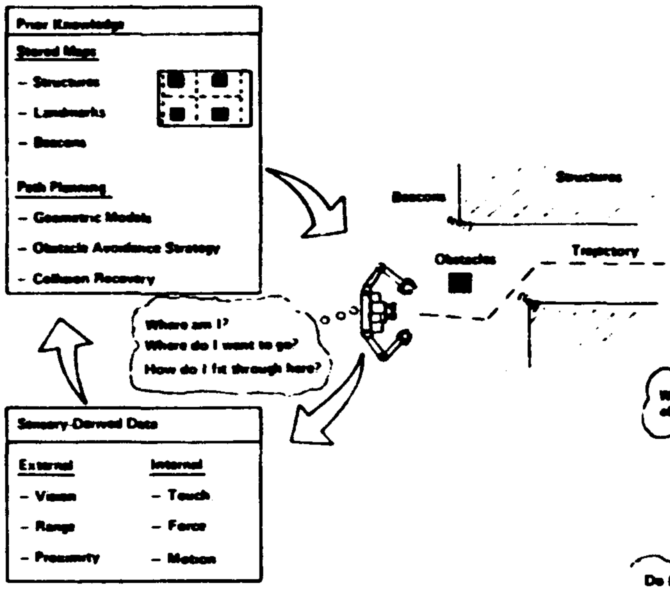


Figure 12. Navigate

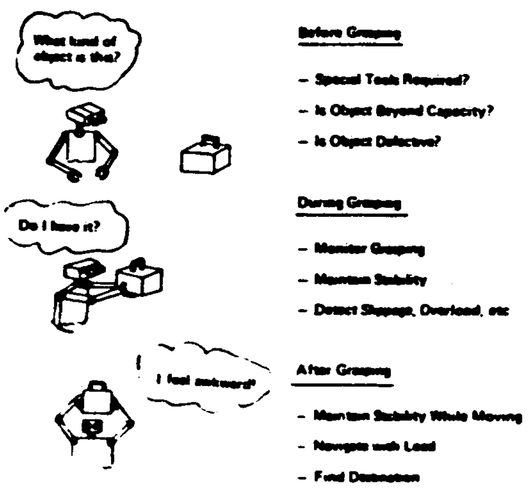


Figure 13. Manipulate

4.3 Manipulate

If the robot is successful in getting to the object (Figure 13), it must, among other things: determine whether neighboring objects must be moved to get at it, inspect the object to see if it is defective, detect the need for special grappling tools, find grasp points, and estimate the weight and strength of the object. After grasping the object, sensors must monitor overall stability and detect slippage and overload. Once the object is securely in the robot's grasp, the robot must navigate to the goal as a different dynamic system which may alter the chosen route. The majority of these subtasks will require the intervention of a human operator.

4.4 Complexity

Even a cursory view of the go-fetch example indicates that there is a tremendous complexity involved. Upon analysis, the various subtasks are interrelated, and the identification of various structures such as hierarchies and feedback loops can only have limited effect, since even they are tangled among each other. Defining "primitive" tasks, such as peg-in-hole and turn-crank, is an excellent way to control the complexity. Although this approach is somewhat limited by the discrete primitives and combinatorial explosion, it represents a solid hierarchical approach that can build up complex tasks from easily understood primitives (see [3] for an example of this approach).

It is not uncommon for a robotics engineer to gradually realize that what at first appeared to be easy to automate is extremely complex, defying all attempts at automation. The history of vision and natural language research provide classic examples, but this phenomenon runs throughout the robotics world.

Humans have an uncanny ability to simplify things. The environment presents an extremely complex array of information. Humans process it, extract the important details, attach symbols, and reason about the symbols. This innate ability to simplify and deal powerfully with abstractions is an advantage, but it also has a tendency to create the illusion that things really are simple. The robotics engineer often finds himself wrestling with exactly this problem. The natural language used in describing everyday tasks does not - and should not - emphasize the complexity involved in automating them. It is up to the robotics engineer to tear down these "simplifying" abstractions to ascertain the primitives and underlying structure that is amenable to automation.

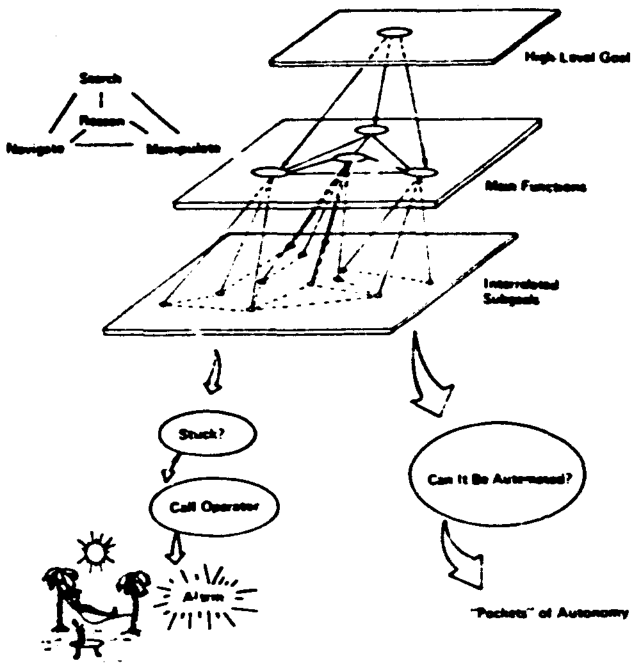


Figure 14. Hierarchy

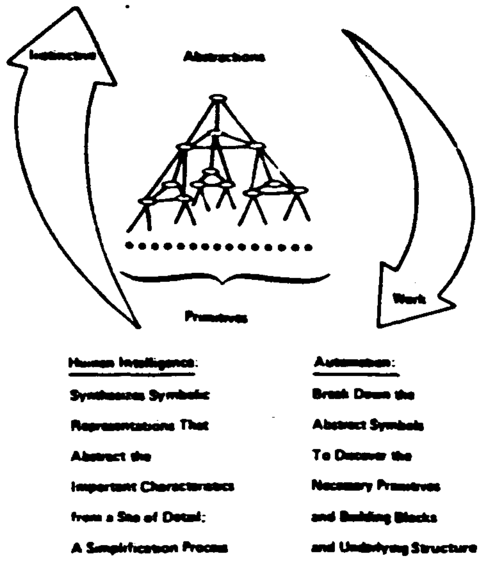


Figure 15. Complexity

4. PARALLELISM IN SUPERVISORY CONTROL SYSTEMS

Real-time execution of distributed supervisory control systems will require implementation on parallel and advanced computer architectures. This requirement is driven by the types of processing and processing requirements in supervisory control systems. As is the case with understanding and describing a problem, it is useful here to think in terms of levels or hierarchies. Since the types of processing in a supervisory control system are so diverse, ultimate implementation will require a network of special purpose, parallel, and serial architectures. In the remainder of this section we briefly consider the types of processing in a supervisory control system and the suitability of various architectures to these levels. Furthermore, we suggest that the biggest problem facing parallel implementation of supervisory control systems is lack of mature software tools and techniques for implementation of these systems on distributed and parallel architectures.

5.1 Processing Requirements

The processing requirements of a supervisory control system can be roughly divided into three levels which suggest the types of architectures which are beneficial. These levels are not necessarily mutually exclusive as tasks may occur which straddle several levels.

5.1.1 Low Level Processing

This level is is connected with interaction with the environment through sensors and commands to effectors and actuators. Typical tasks performed at this level are: sensor data processing/filtering such as image feature extraction (convolutions), associative retrieval, and simple trajectory generation. This level is generally characterized by the following requirements: 1) High bandwidth I/O rates; 2) Numeric processing; 3) Algorithmic processing rather than search among alternates; 4) Identical, repeated operations on bounded regular data structures.

Special purpose hardware is often applied to much of the processing at this level. The special purpose hardware can be serial or parallel, but typically is designed for very specific functions such as convolutions and does not support general purpose processing.

SIMD processors [4] are particularly well suited to this level of processing. Examples of this class of parallel processor are the Illiac IV and MPP [5]. These machines are particularly well suited for problems involving identical operations on bounded regular data structures such as matrix multiplication, parallel sorting, and fast fourier transforms.

MIMD processors [4] such as the BBN Butterfly and Cm* have also been applied to these types of problems, however, they have not achieved the same level of performance as SIMD machines [5,6].

5.1.2 Mid Level Processing

This level concerns more general purpose processing than either the lower or higher levels. This level generally performs operations on filtered, preprocessed sensory data from the lower level. Typical functions performed here are connectivity analysis (object detection), simple vehicle route planning, and optimization problems using techniques such as dynamic programming. This level is characterized by the following requirements: 1) Medium bandwidth I/O rates; 2) Varied operations on regular or irregular data structures containing similar data types; 3) Optimized choice among a bounded set of alternatives.

SIMD machines can be effectively applied to this level of processing if the data to be operated on can be allocated uniformly across the processing elements and the processing performed on the data elements is similar. In the absence of these, many of the elements in the processing array remain idle too much of the time resulting in inefficient use of the system. In such cases, MIMD processors can be applied.

The MIMD, or Multiprocessor architectures, are the most general parallel processors and are capable of executing multiple control threads in parallel. Tightly coupled (shared memory) MIMD computers are applicable when the problem requires a high degree of interprocess communication and a high degree of sharing of objects/resources in the system. These machines can support a finer degree of process granularity due to the high interprocess communication bandwidth (on the order of memory bandwidth). If the problem calls for minimal data/resource sharing, then loosely coupled multiprocessors are applicable and will not be subject to performance degradation due to resource contention as is often the case in a shared memory system.

5.1.3 High Level Processing

Typical functions performed at this level are route and path planning by heuristic search such as A*, mission planning, logical inferencing, and object recognition and understanding. This level operates on data that has been preprocessed by each of the lower levels and typically controls the lower levels. It is characterized by: 1) Low bandwidth I/O rates; 2) Varied operations on different data types and irregular data structures; 3) Symbolic processing; 4) Heuristic search of a large number of alternatives.

MIMD machines are the most widely applied architectures to this level of processing. This is due to the flexibility in these systems since different processors can be applied to different aspects of the problem. Examples are parallel branch-and-bound search and parallel execution of logic programs [7].

5.4 Parallel Programming

The biggest challenge facing the use of parallel processors for supervisory control systems is the complexity of programming many parallel machines. This difficulty arises because programmers must often familiarize themselves with low level details about the parallel architecture as well as parallel programming techniques before they can become effective users of a parallel machine. In order to take advantage of the parallel hardware, conventional programs must often be rewritten and embedded with system calls for memory and process allocation. In addition, when software is ported from one parallel machine to another, it must again be rewritten to account for a different architecture and set of parallel language features. Parallel programming technology is maturing, and advances in the following areas will result in more efficient programming:

1) Optimizing compilers for serial languages. Such compilers will allow for portability of existing codes to parallel machines by making the underlying parallel architecture transparent to the user. Furthermore, this will allow for portability of software across different parallel architectures. Progress in this area has been made for both conventional languages such as C and Fortran [8] as well as AI languages such as functional languages and Prolog [9]. We are currently working on a parallel interpreter for Prolog execution on the Butterfly which automatically optimizes these programs for parallel execution.

2) A common set of parallel programming and language abstractions which can be applied to the various languages and ported across different parallel machines. This coupled with optimizing compilers allows for the use of parallel architectures at multiple levels of expertise. Novices can write programs which are automatically parallelized and optimized for the particular architecture. As they become more experienced and knowledgeable about the particular architecture being used, they can optimize programs even further by embedding these abstractions - or compiler directives - within their programs.

3) A common set of resource sharing and process intercommunication protocols which will allow for programs or processes operating on heterogeneous processors in a network to communicate.

6. CONCLUSION

There are many open issues concerning the development of supervisory systems for Space Station but the key factor is the man-machine interplay. System design methodologies must allow the humans who possess the majority of the intelligence to use the subsystems as tools. The subsystems should amplify human capabilities by allowing the operators to direct, troubleshoot and enhance system performance. From a conceptual standpoint, both hierarchical decompositions and feedback loops are common to man and machine thereby providing a simple framework for interaction. Unfortunately, typical tasks are quite complex, requiring that humans solve them. As experience is gained, the human operators will be in the position to encode new procedures, provided that they have the right development tools.

Parallel implementation of these supervisory control systems will provide the performance necessary to sustain real-time execution of such systems. Ideally, distributed computer systems consisting of serial, special purpose, and parallel processors will allow for optimal performance of the different processing requirements of such systems. The major difficulty, however, in using parallel architectures effectively is the complexity of programming them. Recent and forthcoming advances in parallel software technology will reduce the impact of this problem and allow for parallel implementation of these systems.

7. ACKNOWLEDGMENTS

We would like to express our thanks to other members of the Advanced Automation Technology group for their help in writing this paper: Wendell Chun, John Barnes, Peter VanAlta, John Tietz, Kayland Bradford and Roger Schappell.

8. REFERENCES

[1] Barbera, Fitzgerald and Albus, "Concepts for a Real-Time Sensory Interactive Control System Architecture," Proceedings of The Fourteenth Southeastern Symposium on System Theory, April, 1982.

[2] W. Erickson and P. Cheeseman, "Issues in the Design of an Executive Controller Shell for Space Station Automation," Optical Engineering, November, 1986, Vol. 25, No. 11.

[3] J. Barnes, "A Task-Based Metric for Telerobotic Performance Assessment", Workshop on Space Telereobotics, Jet Propulsion Laboratory, Pasadena, California, January 20-22, 1987.

[4] M. Flynn, "Some Computer Organizations and Their Effectiveness", IEEE Transaction on Computers, C-21, no. 9, Sept. 1972, pp. 948-960.

[5] K. Hwang and F. Briggs. "Computer Architecture and Parallel Processing," McGraw-Hill, 1984.

[6] R. Rettberg and R. Thomas, "Contention Is No Obstacle to Shared Memory Multiprocessing," Com. of the ACM, Dec. 1986, Vol. 29, no. 12, pp. 1202-1213.

[7] B. Wah, G. Li, and C. Yu, "Multiprocessing of Combinatorial Search Problems," IEEE Computer, June 1985, pp. 93-108.

[8] D. Padua and M. Wolfe, "Advanced Compiler Optimizations for Supercomputers," Com. of the ACM, Dec. 1986, Vol. 29, no. 12, pp. 1184-1202.

[9] J. Conery and D. Kibler, "Parallel Interpretation of Logic Programs", ACM Symposium on Functional Programming Languages and Computer Architecture, Oct. 1981.